An Introductory Programming Environment for LEGO® MindStorms™ Robots

Robert W. Hasker
Dept. of Computer Science and Software Engineering
University of Wisconsin - Platteville
Platteville, WI 53818
hasker@uwplatt.edu

Abstract

Introductory programming (CS1) courses must cover a number of difficult topics in a relatively short period of time. Motivating students to learn the material is a key issue. Many have suggested introducing LEGO MindStorms robots so that students obtain hands-on learning with immediate feedback. However, the programming environment provided with the LEGO MindStorms robots is too simplistic for CS1. On the other hand, more full-featured programming environments, especially those based on C and C++, are geared towards more advanced programmers. This paper presents an alternative environment designed specifically for CS1 students learning to program in C++.

Our experience with this environment is reported. Preliminary results suggest that students learn as much as those engaging in traditional programming problems and that the students find programming the robots more interesting. This is in contrast to research which shows that the robots can fail to increase interest and actually reduce how much is learned. It is hoped that this work will lead to a measurable improvement in retention among those interested in computer science and related disciplines.

1 Introduction

Teaching an introductory computer science (CS1) course is a challenge on many fronts: students enter with very different backgrounds in both programming and mathematics, students must learn a number of abstract concepts in a relatively short period of time, and students must finish the course with solid programming skills if they are going to be successful in later courses. Most students find CS1 to be very demanding with many either failing to complete it or deciding to not take any other CS courses.

One way to maintain motivation is to provide interesting problems and problem domains for assignments. Among other approaches, several have turned to the LEGO® MindStorms™ robots [6, 11, 17, 19, 20, 13, 9, 15]. These are readily available, relatively inexpensive, robust systems which can be programmed to solve a wide variety of tasks. The central element is a special "RCX" brick which can store programs, drive motors, and read sensors. This brick is sold in a kit with two motors, two touch sensors, a light sensor, and a collection of more traditional LEGO pieces. The kits can be used to construct small robots which can explore an environment, detect obstacles and lights, and solve simple problems.

The kits also include a programming environment, the Robotics Invention System™ (RIS), which is designed for children about 12 years old. It is graphical, consisting of visual "bricks" which can be attached together to construct complete programs. Most people can learn to construct small programs in a few minutes with a minimal amount of instruction. The language is fairly complete with support for the fundamental control constructs, simple procedures, and variables. However, this language is not appropriate for CS1 courses. Students are unlikely to see the relationship between the visual RIS syntax and more traditional languages. Furthermore, RIS has a number of limitations including

- not allowing more than 30 variables in programs,
- not allowing wait operations to be placed in loops if both depend on the same sensor,
- cumbersome operations to manipulate variables,
- no support for arrays and data types other than integers, and
- not allowing procedures to have parameters.

A different language and environment is needed for CS1.

Potentially suitable environments have been created for a number of languages including Java [16], Ada [6, 8], Lisp [13], Forth [1, 10] and Python [5]. However, these environments will not work for CS1 courses taught in C++. Environments do exist for C and C++ [4, 2], but these are not suitable for CS1 students. This paper discusses the shortcomings of these environments, presents a new environment, and gives preliminary results from using it in a CS1 course.

The preliminary results are particularly interesting because of research showing that the robots reduced exam scores and failed to attract students to fields related to computer science [7]. The authors of this study suggest the problem may be the lack of a simulator to provide better assistance to students while debugging programs. We suggest that another contributer may have been attempting to use the robots for nearly all labs and assignments. Our evaluation, based on using the robots for a limited number of

assignments, suggests that the robots can attract students to computer science and at least not reduce learning in CS1.

2 Existing C-based Programming Environments for LEGO RCX Robots

A large percentage of the LEGO robots are sold to people in their mid-twenties. Among this group, a popular programming environment for LEGO RCX systems is NQC, short for "Not Quite C" [4], developed by Dave Baum. This is a mature system which allows people to write C programs and download them into the robots. NQC itself is command-line oriented, but at least one front end provides a more modern development environment [2]. However, NQC itself has a number of limitations apparently arising from a desire to keep the compiler simple:

- NQC error messages assume strong familiarity with C. Often the response to an invalid program is the message "syntax error."
- There is no support for data types other than `int` and arrays of `int`.
- A number of operations require constant expressions as parameters. For example, constants are required when setting the state of sensors or checking a sensor's status.
- While some sensors are accessed through functions, others are accessed through memory-mapped variables.
- User-defined value-returning functions are not supported.
- Named constants are not supported except through `#define`.

Using NQC in CS1 would require teaching a number of details about C (and C++) that distract from the primary goals of the course. Students would also need to be taught to watch for code which might be legal in C but not in NQC. Even more confusing is that NQC supports a `repeat` keyword for repeating actions that has no parallel in C or C++. NQC would be difficult to use in CS1.

The Bricx Command Center (BricxCC, [2]) fixes some of these issues, providing a potentially more appropriate environment in which to program robots in C++. However, supporting C++ requires installing a subset of Cygwin, a Unix environment for Microsoft Windows™ [3]. This environment is too complex for introductory students.

A popular alternative to NQC is legOS [14, 18]. It uses its own firmware so it can provide support for all of C.[1] However, it is built on top of `gcc`, so the user must install `gcc` and its libraries in addition to legOS. Furthermore, it is relatively unstable. Quoting the `README` file that comes with it, "If things don't quite work, work at it. This is definitely beta." Other disclaimers point out that it was explicitly designed for experienced programmers. Thus legOS is not an appropriate tool for CS1.

[1] A number of sources indicate that legOS supports C++; the documentation indicates otherwise.

A less obvious problem with the above environments is that they present an event-driven model. The user typically defines a collection of tasks to solve a problem. Each task is triggered by one or more sensors. Using multi-tasking in this way is certainly good design, but it presents problems for introductory students because of the complexities of handling concurrency. Even if concurrency issues can be avoided, event-driven programming is significantly different from the more traditional style of programming typically taught in CS1. Students who are struggling with such concepts as decision statements and loops can hardly be expected to simultaneously learn about tasks and event-driven programming. [21] reports successfully introducing the event-driven model in an experimental project, but this project used the original RIS programming environment that makes this model more natural. Attempting to introduce an event-driven model in other programming environments would add a significant burden to already heavily-loaded CS1 courses.

A second problem is that the existing programming environments provide little help in fixing errors. Introductory students who are learning the syntax of `if` and `while` should not be expected to handle such messages as "syntax error" or "operation not allowed." Programming environments designed for professional developers are famous for their vague and often misleading error messages. Classic examples of such misleading errors are "statement missing ;" for

```
cout << "The total cost is " cost;
```

(which is actually missing a <<) and "Lvalue required" for the assignment `xs = ys` where both `xs` and `ys` are arrays.

Thus existing environments are inadequate for use by students in a CS1 course based on C++.

3 Robot Support in HiC

To address these problems, we integrated LEGO Mindstorms support into HiC,[2] a programming environment for a subset of C++ specifically developed for use by introductory students [12]. HiC is available at `http://www.uwplatt.edu/~hasker/hic/` along with instructions for use with LEGO robots. HiC accepts a large subset of C++, the subset typically covered in introductory C++ courses. This subset excludes a number of features because they are dangerous to use at an introductory level, including global variables, `goto` statements, assignment as an expression, and `break` as a way to exit loops. It also excludes features which are best introduced to more mature students: bit manipulation, accessing arrays via pointers, templates, exceptions, static variables, and features which simply provide backwards compatibility with C. In addition, it seeks to encourage a more abstract style of programming, requiring students to use `bool` expressions for tests in `if` and `while`, requiring values returned from functions to be consumed, and disallowing `union`. Thus HiC presents a simpler language to students. However, it does not attempt to solve *all* problems for the student. To encourage students to learn debugging skills,

[2] Alternatively interpreted as "High C," *H*asker's *i*nstructional *C++*, or "hick," a reference to its development amongst cows and cornfields.

it does *not* check for code which is clearly invalid such as semicolons appearing right after `while` loop tests.

Experience has shown HiC provides an environment which allows students to focus more on algorithmic issues rather than C++ minutiae. Students become less frustrated, able to get programs to compile with less help from instructors. Furthermore, instructors spend less time addressing tangential issues such as why a compiler would report "illegal structure operation" for using == to compare to structures. This allows them to spend more time helping students understand core concepts.

As shown in Figure 1, HiC presents a relatively simple integrated development environment. It also

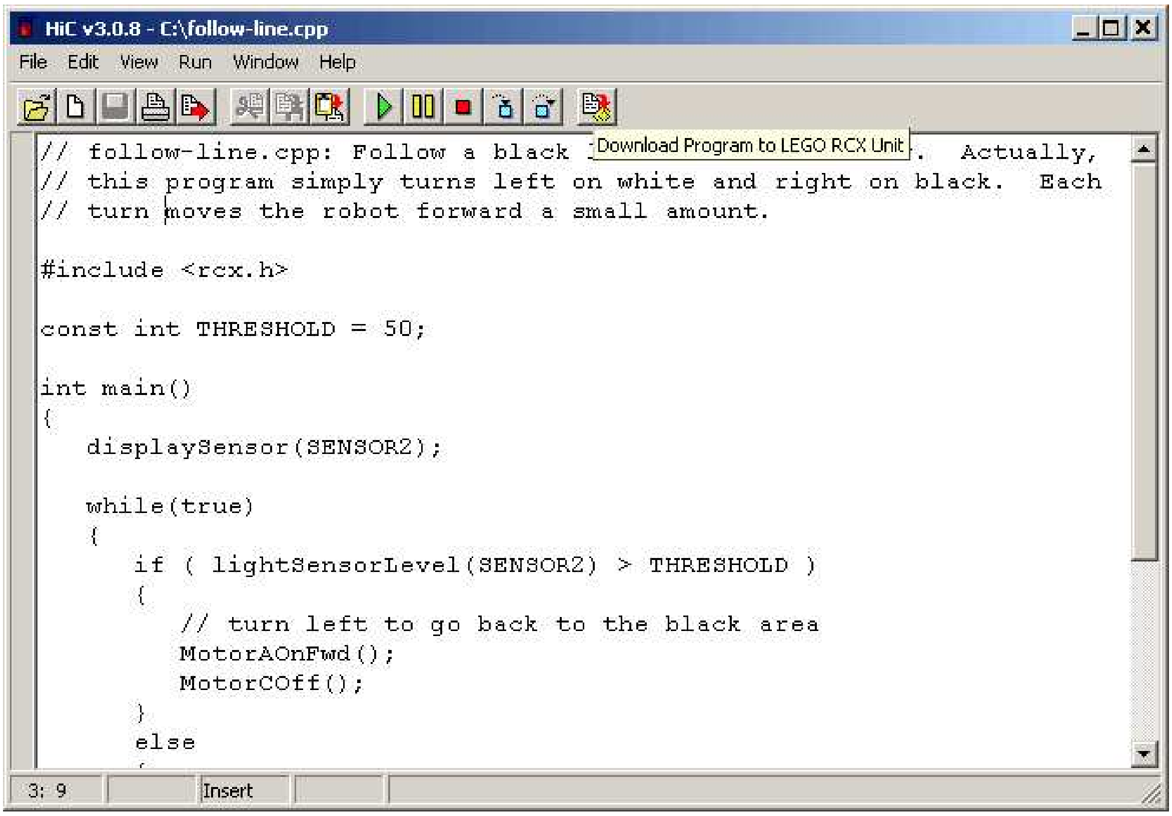


Figure 1: Main HiC window.

provides standard debugging tools for stepping through and examining the internal state of programs. HiC currently runs only on Microsoft Windows. Installing HiC is as simple as dropping the executable on the desktop; all libraries are internal. One of these libraries is "`rcx.h`"; this contains the API for developing programs which control the robot. This API primarily consists of functions to enable and read sensors, control motors, and wait for sensors to be activated.

Students download programs to the robot by simply clicking on the button near the tool tip in Figure 1. HiC then checks that the program does not use any unsupported constructs, translates the C++ source to valid NQC source, and invokes NQC to convert the NQC source into bytecode and download the result into the RCX unit. Both the 1.0 and 2.0 RCX unit versions are supported. While this model means that

a copy of `nqc.exe` must be available, no DLL's or other files need be installed.

Support for LEGO RCX programming in HiC is done through NQC for two reasons. First, it provides a simple interface: all HiC need do is generate a file and invoke NQC to download it into the robot. HiC ensures that the source file is syntactically correct so that the student does not need to interpret NQC error messages. Second, NQC output is based on the same firmware as the RIS programming environment. This simplifies sharing the robots with other courses, such as an introduction to engineering, where the RIS environment *is* appropriate.

The original work to integrate LEGO RCX support into HiC was done by graduate students. This prototype did not extend the language supported by NQC, but it did provide an API which effectively removed the restriction to constant parameters for operations on sensors. Since then support has been added for a number of additional elements: value-returning functions, named constants, `bool`, and enumerated types. Work on adding support for structures, classes, and multi-dimensional arrays is in progress. Support for floating-point types and `std::string` is not planned since these would likely require modifications to the firmware.

4 Experience

The CS1 course at University of Wisconsin–Platteville has no substantial pre-requisite and focuses on procedural programming in C++. Object-oriented methods are taught in later courses. The class meets three times a week for lectures and once for an hour-long closed lab. Each lab is worth 1% of the total grade. The class also includes traditional exams and larger, long-term assignments. All assignments are done using HiC. In spring of 2005 four sections were offered with approximately 20 students in each. These sections were divided into two groups: two of the sections used the robots in some of the weekly labs while the other two sections did not. Students were not told that some sections would be using the robots until after the semester started.

For the closed labs, students are provided with a writeup at the start of the session. The problems are designed to be solved by good students in a few minutes and by all within an hour. A faculty member is available during the lab time to answer questions and otherwise provide help. Students can also help each other, but each student must submit her or his own solution. The labs are "closed" in the sense that students are highly encouraged to show up for the hour, but they can continue to work on the labs afterwards for partial credit.

At the time this paper is being written, six labs have been completed. The LEGO robots were used for two of them. The first time was for lab 3, the first lab involving loops. For the non-robot version of the lab, students were given the start of a program which counted chips with different colors, ostensibly for finding the value of a stack of poker chips. Students completed this program by filling in such things as the loop condition and completing an `if` statement. Students also completed loops and `if` statements for the robotic version of the lab. When complete, the robot program moved the unit forward until it detected a shadow. The `if` statement in this program handled the case of running into objects, making

Section	Count	Quiz 1 & 2		Exam 1		Quiz 3 & 4		GPA	
		Mean	σ	**Mean**	σ	**Mean**	σ	**Mean**	σ
Regular	21	4.22	0.73	47.29	6.02	4.04	0.53	2.55	0.79
Robot	18	4.46	0.48	48.58	4.73	4.18	0.54	2.90	0.64

Table 1: Quiz and exam scores for two sections.

the robot back up and try again to the right. The lab also taught the basics of robot programming since this was not covered in the lectures.

The second lab involving the robot was lab 5, the first lab involving functions and parameters. The robot version of the lab involved moving towards a flashlight, circling to re-obtain the flashlight whenever its light disappeared, and stopping when an obstacle was encountered. The non-robotic lab involved computing prime divisors. In both cases students were given a skeletal program and told what pieces were missing: function calls, function headers, and certain function bodies.

Most students were able to finish the robotic lab in close to an hour. A few took an extra half hour. Traditionally the labs are run by a single faculty member. Because HiC provides reasonably clear error messages and usable debugging tools, a single faculty member can easily answer questions in a timely manner. But using the robots was new, so a least one additional instructor, and sometimes two, provided additional assistance for the robotic labs.

This assistance was important. Few students would have completed the labs without it. The basic problem is the same as alluded to in [7]: programming the robots is difficult because there are few debugging tools. While it might be visually obvious when a solution is not working, using that information to identify the error can be difficult. For example, an infinite loop caused by an errant semicolon merely results in the robot becoming unresponsive. For a traditional problem, a hint could be given that the student should step through the program. When the student discovers the same line is being executed repeatedly, the student can often identify the error without further help. But because HiC does not currently support stepping through a robot program, instructors had to essentially walk the student through the program "by hand" to help them understand the problem.

Work is currently underway to allow HiC to step through robot programs. It is hoped that this will enable students to work somewhat more independently on the labs. If it turns out that additional faculty are always required for robot-based labs, the usefulness of robots will certainly be questioned!

We used two tools to evaluate the effectiveness of the robots. One was performance on quizzes and exams. We took advantage of the fact that two of the sections were taught by the same instructor, organizing things so that one section using the robots and the other not. The statistics for the quizzes and exams for these two sections are given in Table 1. Quizzes 1 and 2 were before the robots were introduced into the labs, exam 1 was given the same day as lab 3 (the first robot lab), and quizzes 3 and 4 were after lab 3 (with quiz 4 after lab 5). In each case, the Student's t test shows the differences between the means was insignificant. In fact, the differences in quizzes 1 and 2—before the robots were introduced to students—suggest that GPA is a much stronger predictor of outcomes. With these sample

Labs	Group	Count	Mean	σ
3 & 5 (robotic)	Control	30	3.233	0.704
	Robotic	22	3.932	0.890
4 & 6 (standard)	Control	29	2.914	0.825
	Robotic	20	3.250	1.060

Table 2: Student interest for the robotic and non-robotic labs.

sizes it is not possible to establish if the robotic labs improved performance, but at least the performance was not measurably worse. Results from the final exam, especially those problems touching on material covered in the robotic labs, will likely provide a more informative comparison of students' performance.

The second tool was an in-class survey. This addressed the question of whether students found the robotic labs interesting. Students were asked to rate each of the labs on a scale from 1 to 5 where 1 was "very low interest" and 5 was "very high interest." To simplify the analysis, each student's ratings for labs 3 and 5 (for which two of the sections used the robots) were averaged together and each student's ratings for labs 4 and 6 (for which no one used robots) were averaged. We then compared the average ratings over both pairs of labs for the group which used the robots against the group which did not. This data is shown in Table 2. The Student's t test shows that the difference of 0.699 in the ratings between the two groups over labs 3 and 5 is significant with a t value of -3.05 and a p value of 0.004 (with 50 degrees of freedom). In comparison, the difference over labs 4 and 6—in which all students did the same problem—was *not* significant: the t value is -1.19 and the p value is 0.241 (with 47 degrees of freedom). This suggests that the preference in labs 3 and 5 *was* particular to those labs.

There are important caveats to this result: while students did not know some sections would use the robots before registering for classes, the sampling was certainly not random. Just as importantly, having additional faculty for the robotic labs may be the reason students preferred those labs. Subjectively, however, it seemed that students were particularly proud of completing the robotic labs.

5 Conclusion

HiC provides a C++-based alternative for programming LEGO Mindstorms robots. The environment is geared towards introductory students, supporting a subset of C++ that is appropriate for CS1. Our experience from using the robots in closed labs suggests that students are interested in working with the robots, though tools need to be developed to allow the students to work more independently. As we gain further experience, we plan to use the robots for open-ended assignments as well.

Previous research [7] showed that basing the large numbers of labs and assignments on the robots actually decreases learning and does not attract students to computer science. It is hoped that taking a less invasive approach does more to encourage students in CS1 courses without negatively affecting learning.

6 Acknowledgments

Thanks to Joe Clifton for writing and for the University of Wisconsin–Platteville for funding a grant to purchase the robot kits. Thanks also to the Joint International Master's students—Andreas Altmannsberger, Christian Döring, Tanja Medschinski, and Volker Schmitt—for the initial work adding LEGO RCX support to HiC. Further thanks to Jim Gast, Qi Yang, and Ambrish Vashishta for allowing their classes to be used for evaluation.

References

[1] `http://www.hempeldesigngroup.com/lego/pbForth/homePage.html`.

[2] `http://bricxcc.sourceforge.net/`.

[3] `http://cygwin.com/`.

[4] BAUM, D. `http://bricxcc.sourceforge.net/nqc/`.

[5] BLANK, D., MEEDEN, L., AND KUMAR, D. Python robotics: an environment for exploring robotics beyond LEGOs. *ACM SIGCSE Bulletin 35*, 1 (January 2003), 317–321.

[6] FAGIN, B. Using Ada-based robotics to teach computer science. *ACM SIGCSE Bulletin 32*, 3 (September 2000), 148–151.

[7] FAGIN, B., AND MERKLE, L. Measuring the effectiveness of robots in teaching computer science. *ACM SIGCSE Bulletin 35*, 1 (January 2003), 307–311.

[8] FAGIN, B. S., MERKLE, L. D., AND EGGERS, T. W. Teaching computer science with robotics using Ada/Mindstorms 2.0. In *Proceedings of the 2001 Annual ACM SIGAda International Conference on Ada* (Bloomington, Minnesota, USA, 2001), ACM Press, pp. 73–78.

[9] FLOWERS, T. R., AND GOSSETT, K. A. Teaching problem solving, computing, and information technology with robots. *Journal of Computing Sciences in College 17*, 6 (May 2002), 45–55.

[10] FRENGER, P. Forth Mindstorms. *ACM SIGPLAN Notices 36*, 12 (2001), 16–19.

[11] GOLDWEBER, M., CONGDON, C., FAGIN, B., HWANG, D., AND KLASSNER, F. The use of robots in the undergraduate curriculum: experience reports. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education* (Charlotte, North Carolina, USA, 2001), ACM Press, pp. 404–405.

[12] HASKER, R. W. HiC: A C++ compiler for CS1. *The Journal of Computing Sciences in Colleges 18*, 1 (Oct. 2002), 56–64.

[13] KLASSNER, F. Using LEGO Mindstorms across the computer science curriculum. *Journal of Computing Sciences in Colleges 18*, 1 (October 2002), 116–116.

[14] LAWHEAD, P. Events robots and programming using Legos in CS1. *ACM SIGCSE Bulletin 33*, 3 (2001), 183.

[15] LAWHEAD, P., DUNCAN, M. E., BLAND, C. G., GOLDWEBER, M., SCHEP, M., AND BARNES, D. J. Legos, Java and programming assignments for CS1. *SIGCSE Bulletin 35*, 1 (June 2003), 47–48.

[16] LAWHEAD, P. B., DUNCAN, M. E., BLAND, C. G., GOLDWEBER, M., SCHEP, M., BARNES, D. J., AND HOLLINGSWORTH, R. G. A road map for teaching introductory programming using LEGO© Mindstorms robots. *SIGCSE Bulletin 35*, 2 (June 2003), 191–201.

[17] LINDER, S. P., NESTRICK, B. E., MULDERS, S., AND LAVELLE, C. L. Facilitating active learning with inexpensive mobile robots. *Journal of Computing Sciences in Colleges 16*, 4 (2001), 21–33.

[18] NOGA, M. L. `http://www.noga.de/legOS/`, 1998.

[19] PATTERSON-MCNEILL, H., AND BINKERD, C. L. Resources for using LEGO® Mindstorms™. *Journal of Computing Sciences in Colleges 16*, 3 (March 2001), 48–55.

[20] SCHEP, M., AND MCNULTY, N. Use of LEGO Mindstorm kits in introductory programming classes: A tutorial. *Journal of Computing Sciences in Colleges 18*, 2 (December 2002), 323–327.

[21] WOLZ, U. Teaching design and project management with Lego RCX robots. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education* (Charlotte, North Carolina, USA, 2001), ACM Press, pp. 95–99.